Unit 4: Loops

In Unit 4 you learned how to make use of **loops** to enter as many values as needed and, as an extension, checks for valid values entered and uses **If** statements to display an appropriate message.

Objectives:

- Try these additional tasks to practice what you learned in Unit 4.

1. Display the powers of a number, like 2, 2^2, 2^3, 2^4, 2^5, … . Input the base and the number of powers to display.
2. Create a countdown timer. Input a number of seconds (like 10) and display the countdown in the center of the screen using **Output( )**. Use the **Wait** statement found on the prgm>CTL menu. **Wait 1** means wait one second before proceeding. At the end of the countdown display "TIME'S UP!". Bonus: enter a number of *minutes* and display the countdown in minutes and seconds (mm:ss) format.
3. You want to average a set of numbers but do not know how many numbers are in the set, but you do know that they are all greater than 0. Use a **While** loop that ends when 0 is entered and keeps a running total and count of the numbers entered. After the **While** loop, calculate and display the average of the numbers. Be careful not to count the 0.
4. **randInt(0,9)→A** produces a random digit (**randInt** is on [math]>PROB). Use a **Repeat** loop to select three random digits that are *all different*. After the loop ends, display the three digits to be sure that they are all different.
5. Write a **Repeat** routine to make sure that three values entered can be the sides of a triangle. You will use this in a later project.
6. Write a program to let the user enter two whole numbers ad computer the sum of the whole numbers between (and including them. Caution: the program must first check to determine which of the two numbers is smaller.
7. **Slope Game:** Make a game that creates 2 random ordered pairs (x,y) using **randInt(-10,10)** so that all four coordinates are integers between -10 and 10. Ask the user to find the slope between the two points. If the user is correct, add a point to a score variable and ask another question. If the user is incorrect, exit the loop and display the number of correct answers. If x1 = x2 in the generated numbers, make x1 = x2 + 1 so you won't have to worry about undefined slopes. Hint: you *can* enter fractions to an **Input** statement.
8. **Perfect Cube Game:** Make a game that generates a random integer from -12 to 12. Display the cube of the number and ask the user to enter the cube root. If the user enters the correct base, tell the user he or she is correct, add a point to a running score then display a new question. If the user's answer isn't correct, exit the loop, display the correct answer and the number of questions answered correctly.

9. **Quadratic Factoring Game**: Make a game that creates random factorable quadratic equations in the form: $x^2 + bx + c$. The game should ask the user to enter both integer solutions to the quadratic. If the user is correct, add one to a score variable and generate a new question. If the user is incorrect, exit the loop and display the number of questions answered correctly. Hint: use **randInt()** to create the roots, then determine the values of a, b and c for display.

10. **Cubic Factoring Game**: Make a game that creates random factorable perfect cube equations in the form: $x^3 + c$. The game should ask the user to enter three integers a,b,d in the form: $(x + a)(x^2 + bx + d)$. If the user's factored form is correct, add one to a score variable and generate a new question. If the user is incorrect, exit the loop and display the number of questions answered correctly. If the user is incorrect, display the correct factored form. Hint: generate a random number and cube it to display $x^3 + c$. Then determine a, b, and d for checking the user's response.

11. Extension: allow other leading term coefficients (other than 1) in #9 and #10.